Synchronizing sound from different devices over a TCP network

Jorge Martin Oliver

Final Year Project - 2014
Computer Science and Software Engineering

NUI MAYNOOTH
Ollscoil na hÉireann Má Nuad

Department of Computer Science
National University of Ireland, Maynooth
Co. Kildare
Ireland

A thesis submitted in partial fulfilment of the requirements for the Computer Science and Software Engineering.
Supervisors: Stephen Brown and Joseph Timoney

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Computer Science and Software Engineering, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____ Date: _____

Abstract

Nowadays, we can send audio on the Internet for multiples uses like telephony, broadcast audio or teleconferencing. The issue comes when you need to synchronize the sound from different sources because the network where we are going to work could lose packets and introduce delay in the delivery. This can also come because the sound cards could be work in different speeds.

In this project, we will work with two computers emitting sound (one will simulate the left channel (mono) of a stereo signal, and the other the right channel) and connected with a third computer by a TCP network. The last computer must get the sound from both computers and reproduce it in a speaker properly (without delay).

So, basically, the main goal of the project is to synchronize multi-track sound over a network.

TCP networks introduce latency into data transfers. Streaming audio suffers from two problems: a delay and an offset between the channels. This project explores the causes of latency, investigates the affect of the inter-channel offset and proposes a solution to synchronize the received channels.

In conclusion, a good synchronization of the sound is required in a time when several audio applications are being developed. When two devices are ready to send audio over a network, this multi-track sound will arrive at the third computer with an offset giving a negative effect to the listener.

This project has dealt with this offset achieving a good synchronization of the multi-track sound getting a good effect on the listener. This was achieved thanks to the division of the project into several steps having constantly a good vision of the problem, a good scalability and having controlled the latency at all times.

As we can see in the chapter 4 of the project, a lack of synchronization over c. 100µs is audible to the listener.

Resumen

A día de hoy, podemos transmitir audio a través de Internet por varios motivos como pueden ser: una llamada telefónica, una emisión de audio o una teleconferencia. El problema viene cuando necesitas sincronizar ese sonido producido por los diferentes orígenes ya que la red a la que nos vamos a conectar puede perder los paquetes y/o introducir un retardo en las entregas de los mismos. Así mismo, estos retardos también pueden venir producidos por las diferentes velocidades a las que trabajan las tarjetas de sonido de cada dispositivo.

En este proyecto, se ha trabajado con dos ordenadores emitiendo sonido de manera intermitente (uno se encargará de simular el canal izquierdo (mono) de la señal estéreo emitida, y el otro del canal derecho), estando conectados a través de una red TCP a un tercer ordenador, el cual debe recibir el sonido y reproducirlo en unos altavoces adecuadamente y sin retardo (deberá juntar los dos canales y reproducirlo como si de estéreo de tratara).

Así, el objetivo principal de este proyecto es el de encontrar la manera de sincronizar el sonido producido por los dos ordenadores y escuchar el conjunto en unos altavoces finales.

Las redes TCP introducen latencia en la transferencia de datos. El streaming de audio emitido a través de una red de este tipo puede sufrir dos grandes contratiempos: **retardo** y **offset,** los dos existentes en las comunicaciones entre ambos canales. Este proyecto se centra en las causas de ese retardo, investiga el efecto que provoca el offset entre ambos canales y propone una solución para sincronizar los canales en el dispositivo receptor.

Para terminar, una buena sincronización del sonido es requerida en una época donde las aplicaciones de audio se están desarrollando continuamente. Cuando los dos dispositivos estén preparados para enviar audio a través de la red, la señal de sonido multi-canal llegará al tercer ordenador con un offset añadido, por lo que resultará en una mala experiencia en la escucha final.

En este proyecto se ha tenido que lidiar con ese offset mencionado anteriormente y se ha conseguido una buena sincronización del sonido multi-canal obteniendo un buen efecto en la escucha final. Esto ha sido posible gracias a una división del proyecto en diversas etapas que proporcionaban la facilidad de poder solucionar los errores en cada paso dando una importante visión del problema y teniendo controlada la latencia en todo momento.

Como se puede ver en el capítulo 4 del proyecto, la falta de sincronización sobre una diferencia de 100µs entre dos canales (offset) empieza a ser audible en la escucha final.

Abstract

TCP networks introduce latency into data transfers. Streaming audio suffers from two problems: a delay and offset between the channels. This project explores the causes of latency, investigates the affect of the inter-channel offset and proposes a solution to synchronize the received channels.

Acknowledgements

I would like to express my deep gratitude to Dr. Stephen Brown and Dr. Joseph Timoney, my research supervisors, for their patient guidance, enthusiastic encouragement and useful critiques of this research work but especially to Dr. Stephen Brown for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously has been very much appreciated.

I would also like to extend my thanks to the technicians of the laboratory of the Computer Science department for their help in offering me the resources in running the program.

Finally, I would like to offer my special thanks to my girlfriend Pilar Onrubia, without her help with the language and her continuous support I would face many difficulties while doing this. I also wish to thank my parents for their support and encouragement throughout my study.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

This project is about the synchronization of multi-track sound. The aim is to record music on two separate devices and to reproduce the sound well synchronized on a third device connected with the other two by a network. In this chapter, an introduction of our project concluding with the purpose of the project is going to be explained.

1.1 GOALS

The main goal of the project is to synchronize multi-track sound over a network.

1.2 MOTIVATION

In a normal situation playing audio in a live studio, you need to deal with a lot of wires with the consequence of supporting the price of them, having several problems of configuration, etc. Thanks to this project, we will have the flexibility of not having to deal with wires, obtaining a compelling reason to use it. Also, achieving this, we can do more than playing sound from different sources, and what is more, we can reuse the project to adapt it to play another kind of information such as the information obtained of a sensor network in order to monitor it.

1.3 METHOD

My work is experimental since the behaviour of streams audio tracks aver the network is difficult to predict and also the effect of the listener. Develop a series of programs to perform a series of experiments in order to measure the behaviour, especially for differences in latency.

1.4 OVERVIEW

Network introduces latency and different devices will experience differences latencies. This offset between two audio channels can degrade the listener's experience.

In this project the causes of latency are investigated and the impact of the offset on a listener is explored.

1.5 REPORT OVERVIEW

The project will cover the latency in audio Ethernet networks using the C programming language creating a TCP connection and using PulseAudio sound server.

CHAPTER 2 BACKGROUND AND PROBLEM STATEMENT

Having seen the introduction of the project, we are going to see the background of the topic with an overview of the situation, what the problem is nowadays and how we solve it.

2.1 INTRODUCTION

Audio transport is one of the most fundamental functional components when we are dealing with multimedia contents. It is relatively easy to send audio such as voice or a predetermined file over Internet, but is not as easy as sending individual tracks of audio, adding all to a consolidated track and just receiving it in the destination device. Here, we find the problem of the synchronization and the latency involved in it. We can distinguish between three important latencies involved in our project: Network latency, Operating System latency and Hardware latency.

In our project, we will have these three kinds of latency every time a device sends audio over the network and another one receives it.

2.2 LITERATURE REVIEW

Nowadays, there are several techniques to synchronise the sound over Internet. As Wireless Audio Sensor Network has become more and more used, there have been a lot of researches about the synchronization of these signals. Some researches have been done on the timestamp mechanism based on time synchronization ignoring propagation delay and many other researches have focused on the synchronization of simple gunshot or scream. However, for the synchronization of intermittent and variation of audio stream, there still exists many challenges. One study done by the University of Beijing [1], propose an effective audio synchronization scheme which, while maintain low energy cost, can synchronise the intermittent audio stream adaptively. Thus, they will obtain audio synchronization without the global clock, which save a very important energy. In the other hand, by introducing a feedback loop mechanism, they can maintain high audio synchronisation fidelity even when the sound strength changes with time and the audio source moves around. This study is more focus on the previous synchronization of the sound after being mixed and played, so this knowledge is very useful to keep experimenting in the project in the future.

We can simplify the task of building an audio network by designing it around one of the many existing communication standards used by the IT networks. Ethernet pops up as a natural choice since it provides the best balance between the high bandwidth (Gigabit version) and cost-effectiveness, compared to other technologies such as MADI (Multichannel Audio Digital Interface).
There is another study made in France [2] that speaks about transporting audio over Ethernet, which is our case, while concluding that using Ethernet for transporting real-time audio information, makes two requirement: or either eliminate the causes of unpredictable behaviour or mitigate them with buffering and retransmission strategies on a very known and mastered time bases. A specific packetization scheme must be proposed when we transmit audio over Ethernet since a normal packetization strategy involves a number of trade-offs. This specific packetization scheme consists of optimizing the bandwidth by maximizing the ratio of payload data to header by using the largest payload of 1500 bytes. But a single audio channel coded on 32 bits packed into such a frame would contain 8 milliseconds of material giving us the inevitability of buffering and so an important delay on the audio path (many tens of milliseconds) which won't be acceptable for live broadcast for example.
This study ends telling that networks performances will continue to increase and soon even a highly reliable wireless multichannel audio transport will be possible (using 802.11ac or SuperWifi).

Another study [3] did a subjective listening test to determine how objectionable various amounts of latency are for performers in live monitoring scenarios. The experiment showed that the acceptable amount of latency can range from 42ms to possibly less than1.4ms under certain conditions.

Sound spatialization for headsets can be based on interaural intensity (IIT) and time differences (ITD) [4]. For this project, we do not focus on the intensity since every devices emits with the same intensity, but not in the same time. Furthermore, the apparent source

position is likely to be located inside the head of the listener, without any sense of externalization. Special measures are needed to be taken in order to push the virtual sound sources out of the head.

Introducing frequency-dependent interaural differences, we can achieve a finer localization of the sound. In fact, due to diffraction the low frequency components are hardly affected by IID, and the ITD is lager in the low frequency range.

We can know what the offset is theoretically obtained for a general incident angle θ by the formula:

$$ITD = ((1.5*\delta)/c)*\sin\theta$$

Where ITD is the interaural time differences, δ is the inter-ear distance in meters, c is the speed of the sound and θ is the incident angle shown in the Figure 1. If the user looks at the middle of the speakers, depends on the side he is, the distance between his place and a fictitious line separating each player is the angle θ.

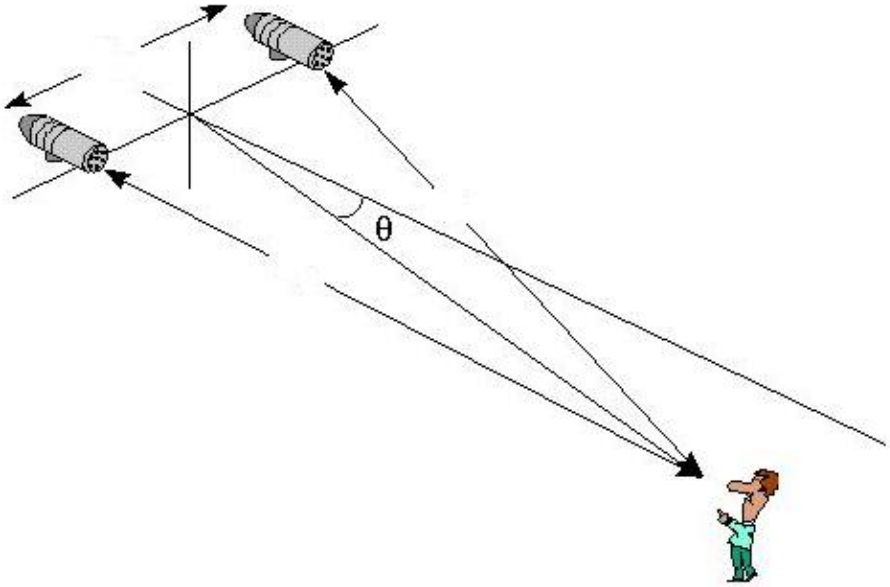

Figure 1. Sound localization geometry

Adding an ITD, if the tracks are not synchronized properly, an artificial apparent movement of the sound source data is introduced.

2.3 PROBLEM STATEMENT

The main problem addressed in the project is the synchronization of the tracks from the different server devices with the client device. Each device may be in different location so each one could cross different switches, hubs or routers adding more latency in one device than in the other when the sound reach the device with the speakers. This means that, in the end device, the multi-track sound will present an offset, that is, the channel will be playing the audio from different sources with a time offset.

The aims of the project are: understanding of the causes, the impact of the listener and to propose a solution.

None of the papers cited above identify the minimum ITD that can be heard a listener. In this project we will do an experiment to find this value.

CHAPTER 3 PROJECT MANAGEMENT

As the problem has been defined and understood in the previous chapter, the management of the project is going to be shown, that is, the method followed to achieve it and how this method was chosen, and how and why was changed during the procedure.

3.1 APPROACH

First of all, I planned the project as a division of several stages instead of doing it in an only step. I decided this because having done it in a big step I would have had serious problems with the implementation of each program, I would have had a lot of errors and, also, I would have dealt with an important latency. Therefore, I planned it in about 8 stages and at the end of each stage I measured the accuracy of the sound synchronization and I fixed it so at the end of each step I had the less latency possible, giving a lot of work done for the next stage.

3.2 INITIAL PROJECT PLAN

As defined below, the initial project plan was following 7 steps until I get the final step, measuring the synchronization accurate of the sound in each step.

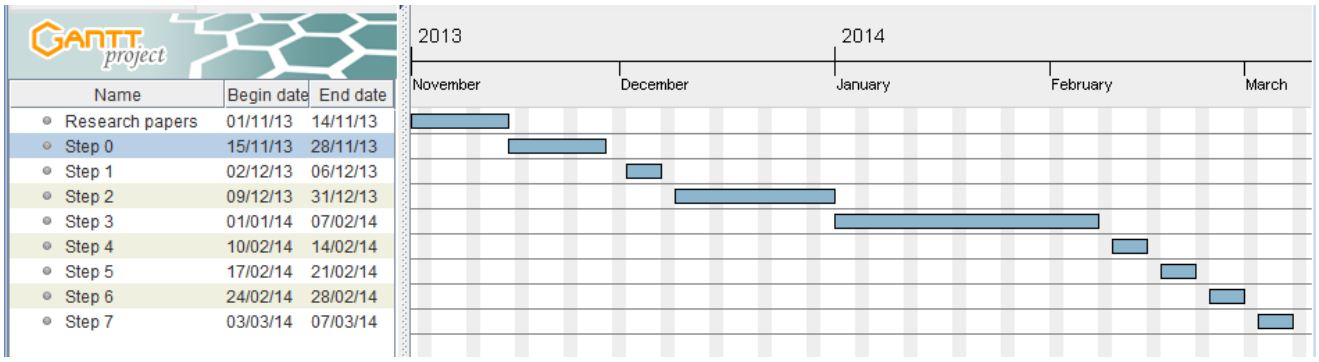

Figure 2. . Gantt diagram with the steps followed in the project

- **Step 0:** Before starting with any step, a program that can record sound from the default input (as a microphone) has to be made/created so this sound could be played then in the speakers. Everything in the same system.

- **Step 1:** Then, the program has to be divided in records (from mic) and mixer/player. In this first step, both the record and the mixer/player are in the same process and in the same system.

- **Step 2:** Now, the record program (recorder) and the mixer will be in different programs but still in the same system.

- **Step 3:** In this step there are two processes for the recorder and another for the mixer. Still everything in the same system.

- **Step 4:** Now, the two processes of the recorders will be in one system and the process of the mixer will be in other. These two systems are connected via Ethernet.

- **Step 5:** In this step, there is one system for each recorder process, and another one for the mixer/player system. Now the three systems are connected via Ethernet.

- **Step 6:** In these last steps, the implementation is finished, and only the systems had to be varied. Now, only the recorder systems are connected via Ethernet, and the last system is connected to them via Wi-Fi.

- **Step 7:** In this last step, the recorder systems are connected separately to the third system (the player/mixer) via Wi-Fi, getting and accurate sound and a barely latency produced by the recorders.

In the figure 3, the steps explained above are explained by images.

3.3 PROBLEMS AND CHANGES TO THE PLAN

Experimenting with the microphone was tedious since a good microphone is required to record the sound with a good quality and the same voice and the same sentence is needed also to try to accurate the synchronization and the quality. To do this, it is also needed a good microphone so the initial plan was changed and instead of using a microphone as the input device to record the sound to reproduce it later, a file is given in the input and the sound of the file is which is reproduced so we will hear always the same sound, we can notice better if the sound is perfectly mixed in the mixer system and hence we can measure the latency easily.

Due to this change, new problems related to the latency could appear such as an offset of the streaming due to the delay of both channels so I made a program to measure the CPU latency and I did a script to know the network latency as well. In the chapter 4.1 a better understanding of these concepts is explained. Thanks to these programs, the experiment of the changing pieces of the code to fight against the latency could be measured and so the solution was reached faster.

Also, I have eliminated the last two steps, in where the connection was done using a Wi-Fi network instead of using an Ethernet network since experimenting with this make you to have two people outside the lab testing and being communicated together and could be very tedious. Anyway having it working in an Ethernet network, should be working via Wi-Fi since the only changes needed are writing the public IP when you call the program instead of the local IP.

3.4 FINAL PROJECT RECORD

So the final project plan is the same as the initial except for once the step 3 was done the step 0 was required to do it again but now using the sound of a file given in the input, and before starting with the 4th step, I did a program that measures the CPU latency and a script that show in seconds the network latency.

The dates do not change even with the changes since it was expected beforehand so for that the step 3 takes one month to be ended.

To conclude, there is an image showing the overall of the project.

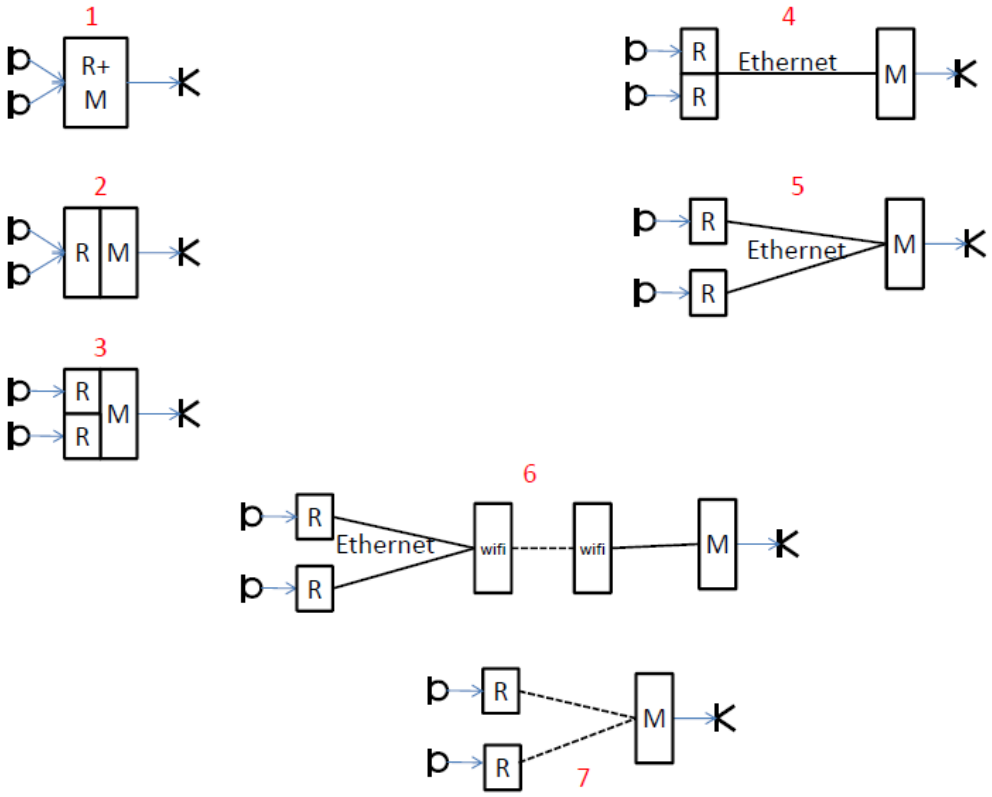

Figure 3. The steps used to carry out the project.

CHAPTER 4 ANALYSIS

In this chapter a deeper analysis of the problem is described.

4.1 PROBLEM MODELLING

As we have seen in the introduction, we need to deal with three kinds of latency in this project.

These sorts of latencies affect to each input channel in a different way so this leads to unsynchronized tracks in the output channel. Therefore, a better understanding of what these latencies are and how to mitigate it are necessary.

When we speak about the Network Latency, we refer to the latency introduced by a packet-switched network. We can measure it in one way (the time from the source sending a packet to the destination receiving it) or in round trip time (the one-way latency from source to destination plus the one-way latency from the destination back to the source) and it depends on the state of our network and it is the easy one to know. With a normal ping command we can measure the round trip time delay. Ping cannot perform accurate measurement but for this case is enough.

Speaking about the Operating System Latency, computers run sets of instructions called a process. In operating systems, the execution of the process can be postponed if other processes are also executing. In addition, the operating system can schedule when to perform the action

that the process is commanding, so the operating system latency is the time taken to the operating system to execute the orders that the program executes. It is the most important latency since it is involved in all the processes of the project. Every time it opens a connection, listens to the device to record the audio, starts the Pulse Audio platform, reads the input given by the device... we can measure a latency which is going to be different depending on the computer, so for this the importance of this latency.

To conclude, there is another latency called Hardware Latency. This delay is narrow related to the latency above since this latency is the delay between the process instruction commanding the transition (operating system latency) and the hardware actually transitioning the voltage from high to low or low to high. In other words, this latency is the tame taken from the call an instruction to the HW actually execute it.

In our project, we are going to deal with these three kinds of latency every time a device sends audio over the network and the third one receives it. The Figure 4 shows our project situation.

Computers A and B are the recorder devices and the computer C is the mixer. The computer A records sound and sends its buffer (buffer A) to the computer C in the time ta over an Ethernet network. The computer B does the same; it sends its buffer (buffer B) but in the time tb. The computer C receives each buffer (buffers A and B) and mixes them having a unique buffer compound of the interleaving of each sample (A|B|A|B...).

In the right side of the image, the time that the computers A and B takes to connect to the client (computer C), listen on the port, initiate the device, the hardware sends the action to start recording and the system really starts to record with how these three latencies are involved in it is shown.

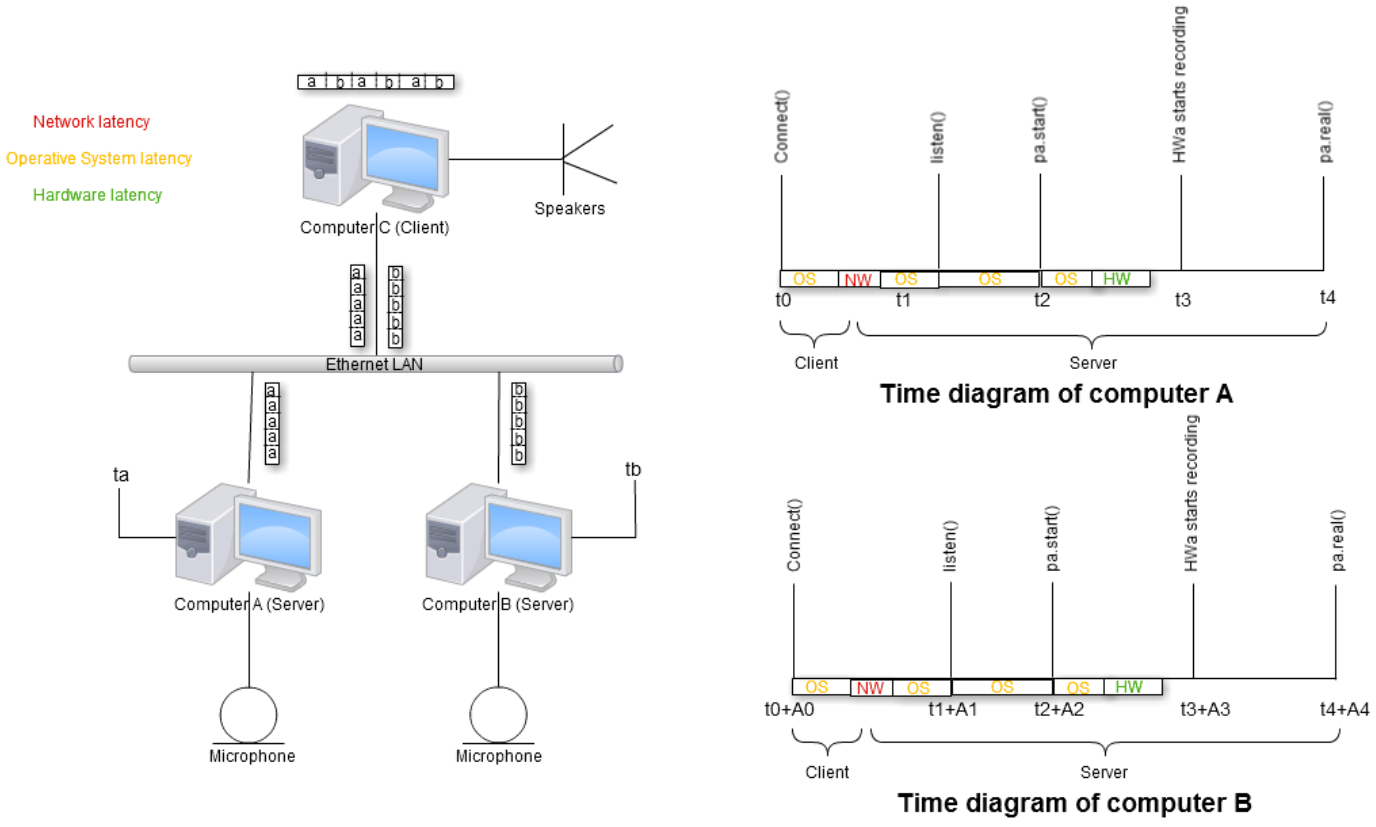

Figure 4. Delay involved in out project

The computer B sends the data at the same time than the computer A (t0), but this with an offset of A0 (t0+A0). This entail a lack of synchronization in the computer C, having to mix

two data arrived at different time with the consequence of an unacceptable synchronization of the sound played in the speakers.

To conclude, knowing the problem that a bad synchronization in the channel can cause, an experiment to learn more about how the offset can affect to the listener was carried out.

This problem consisted on divide a stereo sound track in two mono channels. Then, one was shifted over the other in a range of 10µs to 100µs. The conclusion obtained was that shifting the left channel over the right more than 100µs the ear can notice a lack of synchronization giving the feeling that the sound came from the right. The same conclusion was obtained shifting the right channel over the left, but in this time, the sensation was that the sound came from the left.

The Figure 5 shows the division of the stereo channel in two mono channels and the left channel shifted over the right channel more than 100µs.

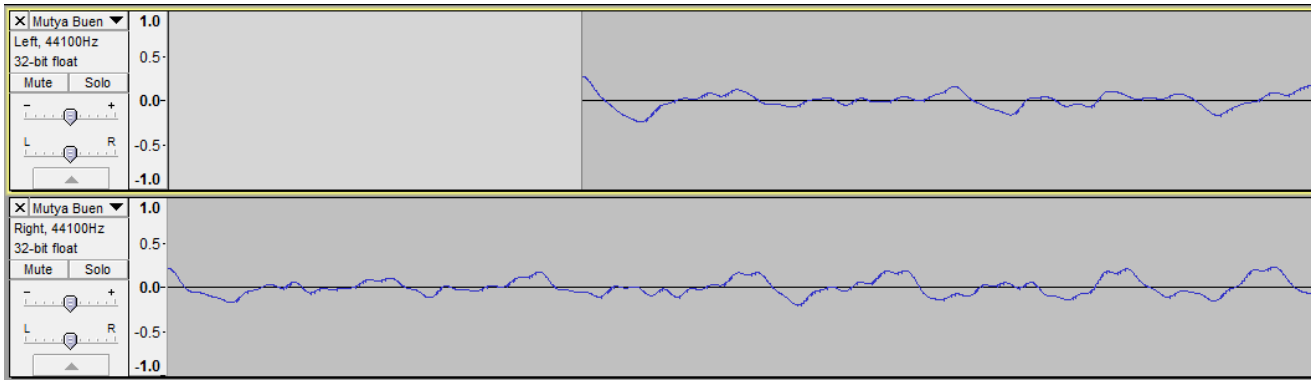

Figure 5. Result of the experiment with the offset.

CHAPTER 5 PRODUCT/SYSTEM DESIGN

Having seen a deep description of our problem, this chapter is describing the design of our system, telling what design was selected, how was selected and why this design was selected instead of others.

5.1 INTRODUCTION

For this project Linux operating system was chosen. Linux is a particularly suitable environment for writing programs and for managing the sound system. This is because, in contrast to some popular proprietary operating systems, it is not necessary to purchase any expensive programming software and, in our case, the appropriate software is already installed on the computer. Moreover, most major distributions of Linux include programming tools on the installation; such tools can be installed very easily at the time of system installation or separately at a later date.

As Linux is the operating system selected and more exactly Ubuntu, the C programming language was chosen to implement all the code needed to deal with the project because it is relatively simple, yet powerful and widely used. In addition, experience with C is useful for obtaining an in-depth understanding of Linux, because they are largely written in C.

Furthermore, it gives us a high level of knowledge of the operating system so any problem could be better scaled.

5.2 INTERFACES TO EXTERNAL HARDWARE AND SOFTWARE

For this system, only headphones, microphones and speakers external hardware are needed.

The microphone is connected in the default input of the device and is needed to record the voice that is going to be stored, processed and sent to the final device. The speakers are connected in the default output of the final device, and are needed to reproduce the sound received and processed beforehand. Finally, headphones are connected in every device in the default output since are very useful to hear easily the interaural difference existing in the sound having more accuracy of the latency between the channels.

As regards to software, the free cross-platform networks sound server PulseAudio was used to design the project. PulseAudio is designed for Linux systems.

PulseAudio is a sound system for POSIX (Portable Operating Systems Interface) OSes (Open System Environments), meaning that it is a proxy for your sound applications. It allows you to do advanced operations on your sound data as it passes between your application and your hardware. Things like transferring the audio to a different machine, changing the sample format or channel count and mixing several sounds into one are easily achieved using a sound server.

To be more exact, the client API for the PulseAudio sound server was used in our devices. The simplified, easy to use, but limited synchronous API was used, since the project was developed in synchronous style and just needed a way to play and record data on the sound server.

The simple API is designed for applications with very basic sound playback or capture needs. It can only support a single stream per connection and has no support for handling of complex features like events, channel mappings and volume control. It is, however, very simple to use and quite sufficient for our programs.

Signed 16 Bit PCM, little endian sound data format was used to manage the sound samples (little endian systems are those in which the least significant byte is stored in the smallest address).

The first step before using the sound system is to connect to the server. This is done by using **pa_simple_new()** function. The lines of code that gets this are in the Figure 6.

```
/* The Sample format to use */
static const pa_sample_spec ss = {
    .format = PA_SAMPLE_S16LE,
    .rate = 44100,
    .channels = 2
};

pa_simple *s_in, *s_out = NULL;
int error;

/* Create a new playback stream */
if (!(s_out = pa_simple_new(NULL, argv[0], PA_STREAM_PLAYBACK, NULL, "playback", &ss, NULL, NULL, &error))) {
    fprintf(stderr, __FILE__": pa_simple_new() failed: %s\n", pa_strerror(error));
    goto finish;
}

if (!(s_in = pa_simple_new(NULL, argv[0], PA_STREAM_RECORD, NULL, "record", &ss, NULL, NULL, &error))) {
    fprintf(stderr, __FILE__": pa_simple_new() failed: %s\n", pa_strerror(error));
    goto finish;
}
```

Figure 6. Code used for connect to the sound server

At this point a connected object is returned, or NULL if there was a problem connecting.

Once the connection is established to the server, data can start flowing. Using the connection is very similar to the normal read() and write() system calls. The main difference is that they're called **pa_simple_read()** and **pa_simple_write()**. It is important to note that these operations always block.

In the Figure 7, the program execute **pa_simple_read()** to read the data recorded with the microphone, and **pa_simple_write()** to play this data into the speakers.

```
/* Reads data from the input*/
if (pa_simple_read(s_in, buf, sizeof(buf), &error) < 0) {

    fprintf(stderr, __FILE__": read() failed: %s\n", strerror(errno));
    goto finish;
}

/* ... and play it */
if (pa_simple_write(s_out, buf, sizeof(buf), &error) < 0) {
    fprintf(stderr, __FILE__": pa_simple_write() failed: %s\n", pa_strerror(error));
    goto finish;
}
```

Figure 7. Code used for read data and write it to the speakers

Now, you can have a control of the buffer used in the data transfer by using **pa_simple_get_latency()** which returns the total latency of the playback or record pipeline, respectively.

```
#if 1
        pa_usec_t latency;

        if ((latency = pa_simple_get_latency(s_in, &error)) == (pa_usec_t) -1) {
            fprintf(stderr, __FILE__": pa_simple_get_latency() failed: %s\n", pa_strerror(error));
            goto finish;
        }

        fprintf(stderr, "In:  %0.0f usec    \r\n", (float)latency);

        if ((latency = pa_simple_get_latency(s_out, &error)) == (pa_usec_t) -1) {
            fprintf(stderr, __FILE__": pa_simple_get_latency() failed: %s\n", pa_strerror(error));
            goto finish;
        }

        fprintf(stderr, "Out: %0.0f usec    \r\n", (float)latency);
#endif
```

Figure 8. Code used for know the latency of the playback

If a playback stream is used then **pa_simple_drain()** operation is available. This will wait for all sent data to finish playing.

```
/* Make sure that every single sample was played */
if (pa_simple_drain(s_out, &error) < 0) {
    fprintf(stderr, __FILE__": pa_simple_drain() failed: %s\n", pa_strerror(error));
    goto finish;
}
```

Figure 9. Code used for wait until all the data is played

Finally, once playback or capture is complete, the connection should be closed and resources freed. This is done through **pa_simple_free(s)**.

```
if (s_in)
    pa_simple_free(s_in);
if (s_out)
    pa_simple_free(s_out);
```

Figure 10. Code used for close and free the resources

CHAPTER 6 SOFTWARE DESIGN

Having seen the design of the system, the software is going to be explained starting with an introduction telling how is the design created and why it was chosen, and finishing with an explanation in a high level showing explanatory images that say how every program done works.

6.1 INTRODUCTION

The design was created increasingly starting with simple programs which record the sound from the input or from a file given and reproduces it in the speakers, then separating the recorder process in one program and the player in another connected via sockets, then dividing the recorder in two separate recorders reading a different channel each one and now the player takes the role of a mixer (mix the sound received from the different recorder) and of a player, playing the sound mixed in the speakers. Finally, the design ends with a different program for each recorder and another for the mixer/player connected via sockets.

All the code used for the design was written in C programming language using the "gedit" text editor. This is the default text editor for GNOME desktop environment; this text editor also emphasizes simplicity and eases of use and was designed to have a clean and simple graphical user interface (according to the philosophy of GNOME). For these reasons this editor is suitable for the project.
To conclude, Audacity software was used to test with the sound once it was played to see with details the delay between both channels and the accuracy of the programs.

6.2 PROGRAMS USED IN THE PROJECT

- Program which reads data from the default input (microphone) and then plays it the speakers (see Figure 3 point 1).

  The Figure 11 shows a loop that once the connection to the sound server is done with the consequence of the creation of a new playback stream, reads data from the microphone and then writes it into the speakers until the user stops the program (normally typing "Ctrl+C").

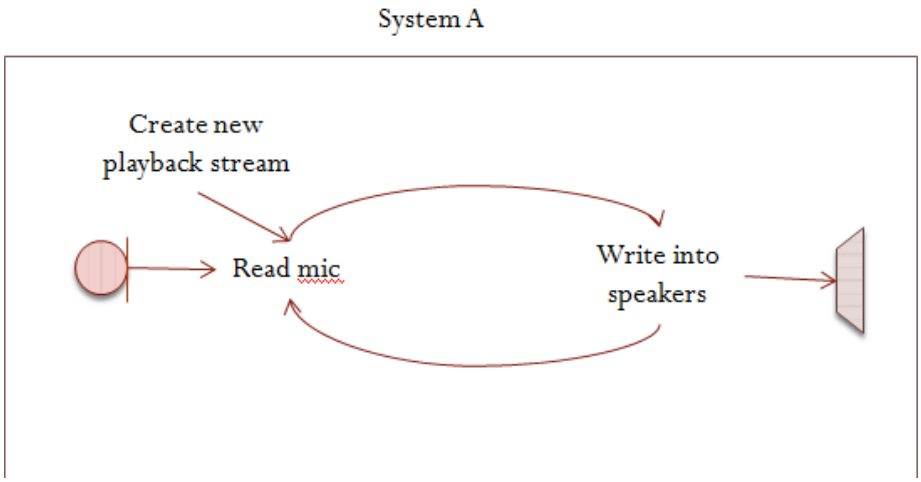

Figure 11. Graphic explanation of the loop

The loop is executed thanks to the **for(;;)** command. The next code is used to execute this loop. The explanation if this code is in the chapter 5.2. Every program explained here will have a similar code to execute this loop.

```
for (;;) {
    uint8_t buf[BUFSIZE];

    /* Reads data from the input*/
    if (pa_simple_read(s_in, buf, sizeof(buf), &error) < 0) {

        fprintf(stderr, __FILE__": read() failed: %s\n", strerror(errno));
        goto finish;
    }

    /* ... and play it */
    if (pa_simple_write(s_out, buf, sizeof(buf), &error) < 0) {
        fprintf(stderr, __FILE__": pa_simple_write() failed: %s\n", pa_strerror(error));
        goto finish;
    }
}
```

Figure 12. Code used for the loop

- Program which reads data from a file and then plays it in the speakers.

  The Figure 13 shows a loop that reads a file and writes the data into the speakers until the file is completely read.

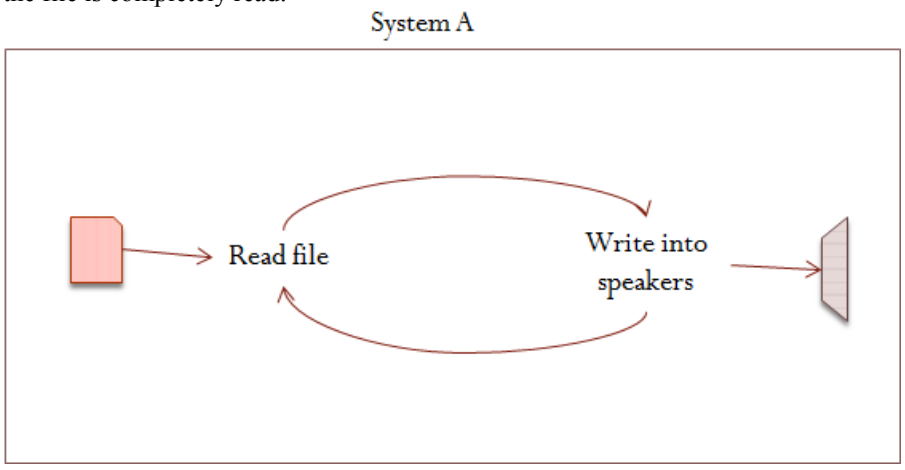

Figure 13. Graphic explanation of the loop

- Program which reads data from two microphones then mixes it and reproduces it in the speakers (see Figure 3 point 3).

  The Figure 14 shows a loop that reads data from two different microphones, mixes it and writes the data into the speakers until the user stops the program.

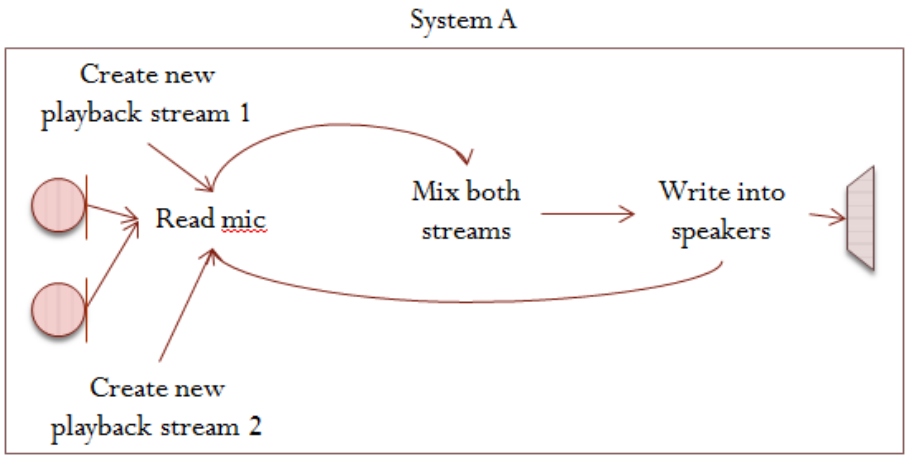

Figure 14. Graphic explanation of the loop

- Program that reads data from the default input (microphone) and then stores it in a file.

  In the Figure 15, a loop that reads continually data from the microphone and writes it into a file until the user stops the program is showed.

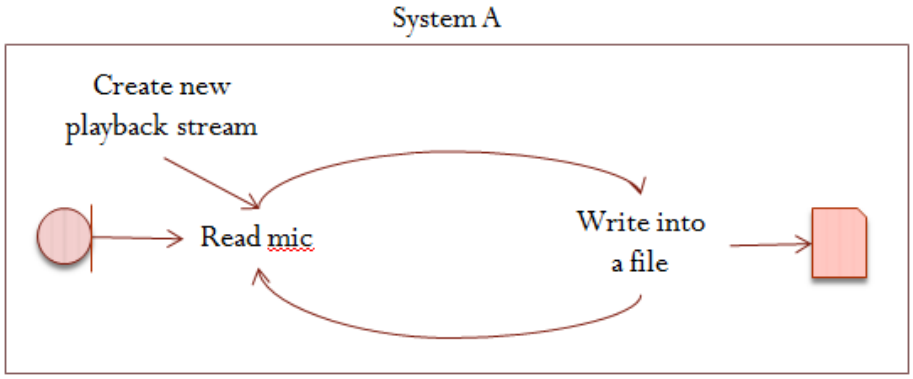

Figure 15. Graphic explanation of the loop

- This program is the same as the first program except for now, it divides the recorder process and the player process in two different systems connected to each other by sockets (see Figure 3 point 2).

  In the Figure 16, the system A reads data from a file, then prepare the sound obtained from this file to be sent, open a socket and wait for a connection. The system B establishes a connection with the system A, gets the sound from the socket and reproduces it in the speakers. The system A is sending data until the user stops it or the system B stops.

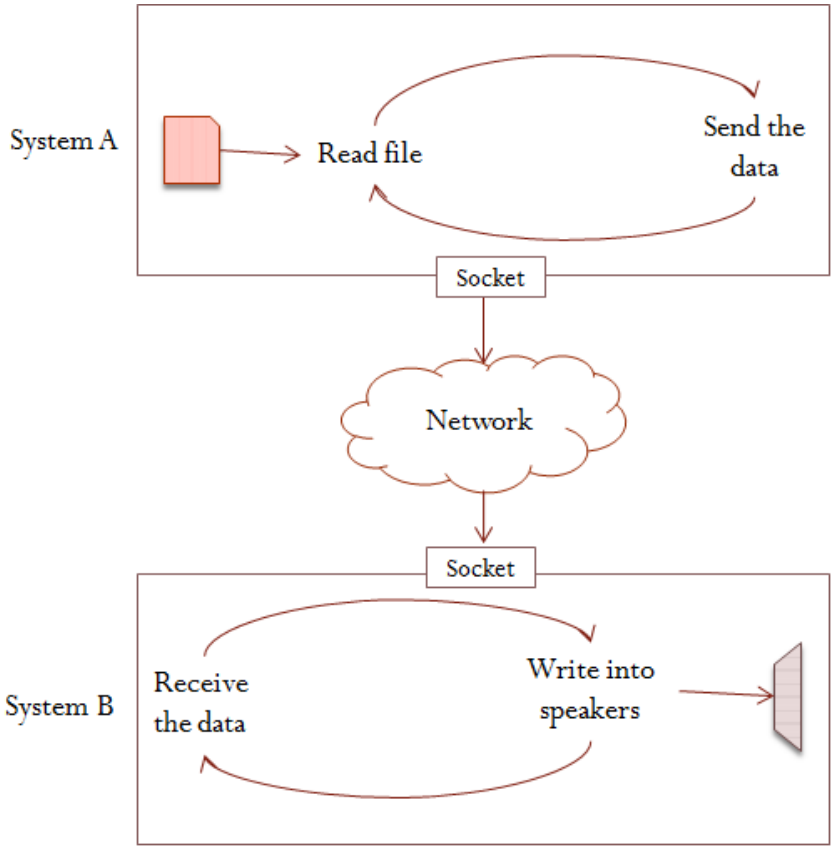

Figure 16. Graphic explanation of the loop

- This program is the same as the previous program, but now, the recorder process is divided in two processes, having three processes in total (see Figure 3 point 5).

  Basically, the loop of the systems A and B programs reads data from the file, store it, open a socket and send it over the network only the even and the odd samples respectively to the third system. The system C opens a socket as well and connects to both systems. The loop of this system receives the data, mixes it and writes the result in the speakers. The system A or B is sending data until the user stops it or the system C stops.

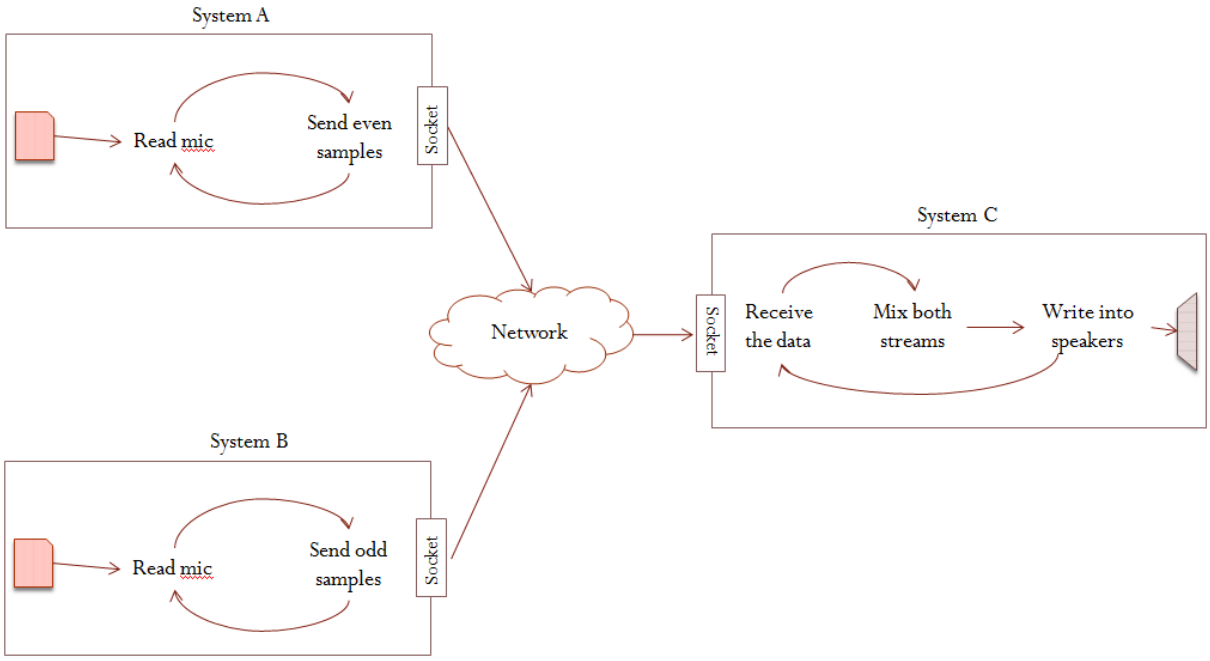

Figure 17. Graphic explanation of the loop

The next two programs are the same as these two previous programs but instead of using a file, these two uses a microphone to record the data. Anyway, explanatory images are going to be show for each program.

- This program is the same as the fifth program (the previous of the previous) but, instead of using a file to record the sound, it uses the sound recorded from a microphone to get the data. It also uses sockets to connect the recorder and the player. It is also divided in two programs/processes.

  In the Figure 17, the system A reads data from the microphone, then prepare the sound obtained to be sent, open a socket and wait for a connection. The system B establishes a connection with the system A, gets the sound from the socket and reproduces it in the speakers. The system A is sending data until the user stops it or the system B stops.

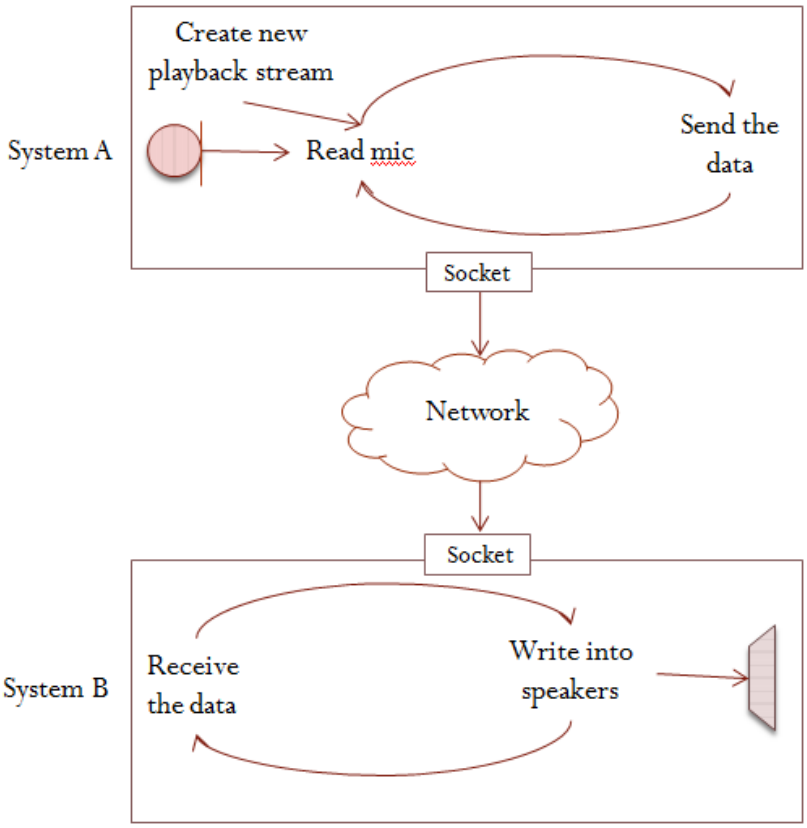

Figure 18. Graphic explanation of the loop

- The last program, is the same as the previous program, but now, the recorder process is divided in two processes, having three processes in total.

  Here, the loop of the systems A and B programs reads data from the microphone, store it, open a socket and send it over the network only the even and the odd samples respectively to the third system. The system C opens a socket as well and connects to both systems. The loop of this system receives the data, mixes it and writes the result in the speakers. The system A or B is sending data until the user stops it or the system C stops.

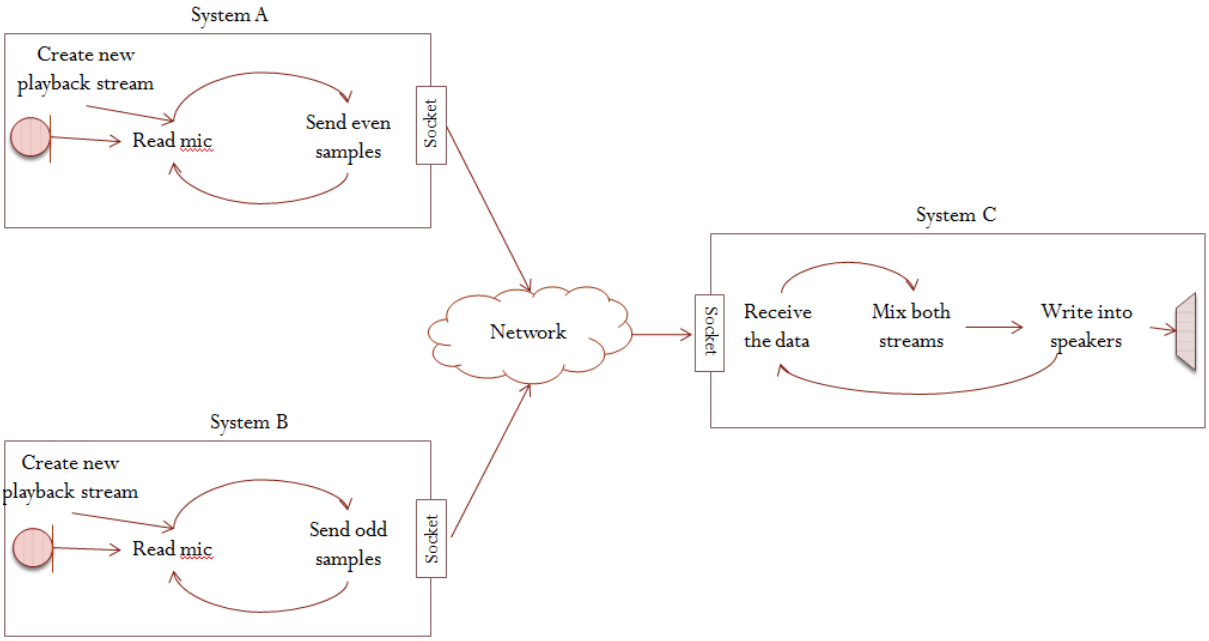

Figure 19. Graphic explanation of the loop

6.3 MIXING

In the mixer program, the process to connect it to both recorder systems is the next: It starts opening the socket with the first system, handles errors and receives the buffer with the even samples of the sound recorded. Then open the second socket, handles errors and receives the second buffer with the odd samples of the sound recorded in the second system. Finally, it mixes both buffers as explained in the next chapter in a third buffer and sends it to the speakers to be played.

The key operation for synchronizing the channels is he mixing described in detail in chapter 7.

CHAPTER 7 IMPLEMENTATION

In this chapter, the verification and the validation of the code is going to be explained in such way that the reader could verify the results of the project.

7.1 INTRODUCTION

As it was told in the previous chapter, the programming language chose is C to be programmed in a device with Ubuntu operating system using “gedit” text editor.

7.2 CODING

Since the final program is huge, before testing the action of mixing both buffers that arrive from the different devices and having several errors without knowing exactly where they came from, a mixer separately program was done.

The program consists of a function that receives two different buffers (stream 1 and 2) and then mixes them having a final buffer (result) with the content of both buffers already mixed.

What the function does is to loop through the two streams storing the stream 1 in the even positions (0, 2, 4, 6...) of the result buffer, and the stream 2 in the odd positions (1, 3, 5, 7...) of the result buffer.

The Figure 20 shows the mixer function.

```
void mix(unsigned short stream1[], unsigned short stream2[], unsigned short result[], int len)
{
        int j;
        for (j=0;j<(len/2);j++){
                result[2*j]=stream1[j];//Even samples
                result[(2*j)+1]=stream2[j];//Odd samples
        }
}
```

Figure 20. The mixer function

In order to synchronize the channels, an offset would be introduced into this algorithm (e.g., Result[2*j] = stream1[j + offset]; Result[(2*j) + 1] = stream2[j].). Note that only one stream is affected with the offset.

7.3 VERIFICATION

For the experiment, three tests were made to verify the code of the previous chapter. One buffer was filled with all ceros and the other was filled with all ones for the first test. After, the opposite, the first was filled with all ones and the other was filled with all ceros giving us the result of the second test. Finally, in the third test, to have a more real case, both were filled with all the values randomly. The result of this mix was successfully as we can see in the Figures below.

Several images with the results of the test:

- **Test 1:** The stream 1 is filled with ten 1s and the stream 2 is filled with ten 0s, giving a result of an array of 20 positions with ones and zeros interleaved with success.

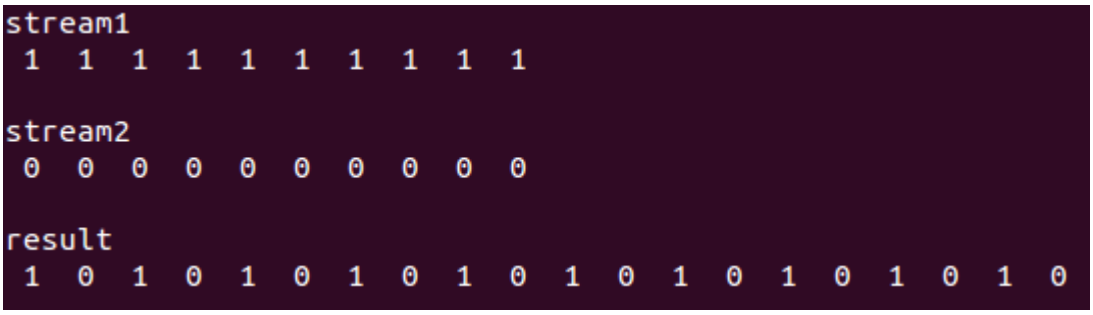

Figure 21. Result of first test

- **Test 2:** The stream 1 is filled with ten 0s and the stream 2 is filled with ten 1s, giving a result of an array of 20 positions with zeros and ones interleaved with success.

```
stream1
 0  0  0  0  0  0  0  0  0  0

stream2
 1  1  1  1  1  1  1  1  1  1

result
 0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
```

Figure 22. Result of second test

- **Test 3:** The stream 1 and the stream 2 are filled with ten random numbers, giving a result of an array of 20 positions with the random numbers interleaved with success.

```
stream1
 17767  9158  39017  18547  56401  23807  37962  22764  7977  31949

stream2
 22714  55211  16882  7931  43491  57670  124  25282  2132  10232

result
 17767  22714  9158  55211  39017  16882  18547  7931  56401  43491  23807  57670  37962  124
 22764  25282  7977  2132  31949  10232
```

Figure 23. Result of third test

7.4 VALIDATION

In every program done, there is little extra code to show the user the time that the program takes to send and receive the buffers of the sound in order to have an idea of how latency involves each program. Then, if the latency takes more time than necessary, changes in the program must be taken to reduce the delay. Therefore, the final program (which receives both streams audio from different devices and then mixes it another buffer to after reproduce it in the speakers) has this piece of code as well, having the less latency possible.

As the chapter 4.1 explain, there are three latencies involved in this project: Hardware Latency, Network Latency and the Operating System latency.

To measure the hardware latency, the **pa_simple_get_latency()** function has been used in the programs. A more detailed explanation of how this function works is given in the chapter 5.2.

To be sure that the program is going to respond with the less latency possible related to the network latency and the CPU latency (operating system), a program that measures both latencies was needed. That way, we can measure the delay produced by this cases, and thus the final latency can be better evaluated.

The CPU latency program, execute a loop 100000000 times and shows the maximum and the minimum time taken to execute one loop in nanoseconds, and the total average in seconds, having an idea of how much time the CPU spends to execute one loop.

```
The loop has been done 100000000 times.
The maximum time = 32960.
The minimum time = 23
The total average time = 23=  0.000000023
```

Figure 24. Result of CPU latency

The network latency is known using the ping command. Although ping cannot perform accurate measurement, for this case is enough to measure the round trip time delay of our connections.

To know the network latency, the ping command was used, giving us an approximate round trip times (rtt) in milliseconds (ms): Minimum (min), Average (avg), Maximum (max) and the Standard Deviation (mdev).

```
rtt min/avg/max/mdev = 0.223/0.260/0.481/0.043 ms
```

Figure 25. Result of executing the command ping

CHAPTER 8 EVALUATION

In this chapter, a list of every program with its experiments and results obtained is going to be showed, giving the reader a deeper understand of how a program was tested to satisfy the problem of the synchronization.

8.1 PROGRAM 1

This program reads data from the default input (microphone) and then plays it the speakers.

- ***Experiments:*** A few lines of code were introduced to show the latency due to the uses of the devices (the microphone and the speakers). This delay is the last latency needed before calculating the final latency since the network and the CPU latency were calculated in the previous chapter. To start this experiment, a person was speaking with the microphone until the user typed "Ctrl+C" to exit the program.

- ***Results:*** The result obtained was the time taken by the devices in microseconds. The Figure 26 shows the time taken to the program to manage the devices (HW) every time the sound was sent to the speakers (T1, T2…). The table of the Figure 26 only shows five samples since is more than sufficient to see how this latency involves in our system, having a maximum of 352395µs in the input and 130210µs in the output leading to the conclusion that this time barely affects to the synchronization.

	T1	T2	T3	T4	T5
Input (mic)	352193µs	352395µs	351707µs	352087µs	93460µs
Output (speakers)	0µs	169µs	7767µs	100849µs	130210µs

Figure 26. Table with the results of the experiment

8.2 PROGRAM 2

This program reads data from a file and then plays it the speakers.

- ***Experiments:*** Here a few lines of code to show the latency due to the uses of the devices were also introduced but, in this case, only the latency due to the speakers was needed. For this experiment, a file with raw data was given as the input to be reproduced in the speakers.

- ***Results:*** The result obtained was the time that the program needs to handle the speakers in microseconds. In this program, we can see a maximum output time of 229700µs concluding that also, this latency barely affects to the synchronization of the sound.

	T1	T2	T3	T4	T5
Output (speakers)	229700µs	200458µs	174525µs	167345µs	208137µs

Figure 27. Table with the results of the experiment

8.3 PROGRAM 3

This program reads data from two microphones then mixes it and reproduces it in the speakers.

- ***Experiments:*** In this program, the lines of code calculate the latency produced to manage both microphones devices as well as by the speakers' device. For this experiment, a person was required to speak in both microphones until the user type "Ctrl+C" to exit the program.

- ***Results:*** The result obtained was the time that the program needs to handle the speakers and both microphones in microseconds. The Figure 28 shows now the hardware latency of the three devices. In this case, as it still is in the same system, the three latencies in each time must be added together, having now a total latency of 742183µs in the last case (T5), feeling know a slightly lack of synchronization in the end.

	T1	T2	T3	T4	T5
Input_a (mic1)	312675µs	312770µs	312863µs	312944µs	313031µs
Input_b (mic2)	312695µs	312782µs	312888µs	312967µs	313053µs
Output (speakers)	23219µs	46439µs	69659µs	92879µs	116099µs

Figure 28. Table with the results of the experiment

8.4 PROGRAM 4

This program reads data from the default input (microphone) and then stores it in a file. This program is useful to test the file recorded with the program that uses a file to record the sound.

- ***Experiments:*** In this program, the lines of code calculate the latency produced to manage only the microphones device. For this experiment, a person was required to speak in the microphones until the user type "Ctrl+C" to exit the program.

- ***Results:*** The result obtained was a file with the sound stored. The Figure 29 shows the content of this file opened with Audacity.

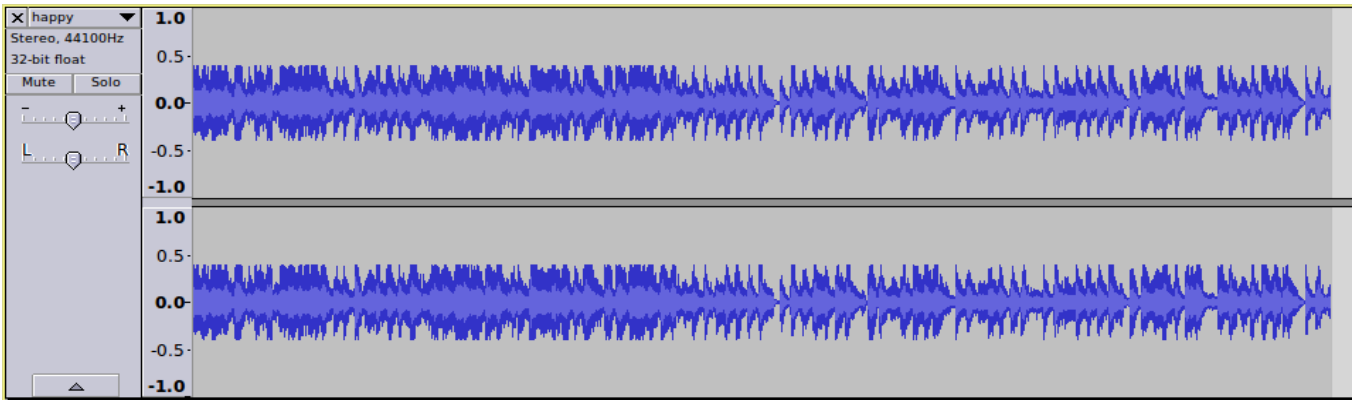

Figure 29. Results of the experiment of the forth program

8.5 PROGRAM 5 (USING A FILE AS THE INPUT)

This program starts using sockets to connect the recorder and the player. This is divided in two programs/processes. The first reads data from a file, then prepare the sound obtained from this file to be sent, open a socket and wait for a connection (server). The second establishes a connection with the server (client), get the sound from the server and reproduce it in the speakers.

- ***Experiments:*** In this program, the lines of code calculate the latency produced to manage only the speakers' device since a microphone device is not needed. For this experiment, a file with raw data was given as the input in the first program and the second program just reproduces in the speakers.

- ***Results:*** The result obtained was the listen of the sound of the file in the speakers with a slightly delay.

  This figures bellow shows the hardware latency between recording + playing single channel (using **pa_simple_get_latency()** function). As we can see, the recorder program (server) works faster than the player program (client), having received only two buffers in spite of the recorder have sent eight buffers.

```
Streaming data server running. Use ^C to exit.
Waiting for a connection...
```

Figure 30. Recorder program waiting for a connection

```
Streaming data server running. Use ^C to exit.
Waiting for a connection...
accepting connection from 127.0.0.1
[12285] Sending buffer 0 [starting at sample 16784]
SIOUTQ=67136
[22160] Sending buffer 1 [starting at sample 33568]
SIOUTQ=67136
[21494] Sending buffer 2 [starting at sample 50352]
SIOUTQ=134272
[21826] Sending buffer 3 [starting at sample 67136]
SIOUTQ=134272
[22378] Sending buffer 4 [starting at sample 83920]
SIOUTQ=201408
[19854] Sending buffer 5 [starting at sample 100704]
SIOUTQ=201408
[17648] Sending buffer 6 [starting at sample 117488]
SIOUTQ=268544
[16205] Sending buffer 7 [starting at sample 134272]
SIOUTQ=335680
[18244] Sending buffer 8 [starting at sample 151056]
SIOUTQ=402816
```

Figure 31. Recorder program sending audio over the network

```
Streaming data client running. Use ^C to exit.
Connected...
Output(speakers): 0 usec
[18415] Playing buffer 0 [start sample=0][size=67136]
Output(speakers): 230297 usec
[169887] Playing buffer 1 [start sample=16784][size=67136]
Output(speakers): 253923 usec
[361763] Playing buffer 2 [start sample=33568][size=67136]
^C
```

Figure 32. Player program connected with the recorder and receiving the sound buffer by the time it is reproducing it

8.6 PROGRAM 6 (USING A FILE AS THE INPUT)

This program is the same as the previous program, but it divides the record process in two processes, having two programs for the recorder part (two servers) and another one that establishes a connection with both servers (client), gets both sounds, mixes it and reproduces it in the speakers.

- ***Experiments:*** The experiments done for this program were the same as in the previous program, but here two recorder programs were needed, so each program reads a file with raw data and prepare the data to be sent to the third device. Thus, the third program just connects to the recorder to start to reproduce it in the speakers.

- ***Results:*** The result obtained was the listen of the sound of the file in the speakers with a slightly delay again.

This figures bellow shows the hardware latency between recording + playing single channel (using **pa_simple_get_latency()** function). As we can see, the recorder programs (server) works faster than the player program (client) having received only four buffers of each server in spite of the recorder have sent twenty-five buffers. Obviously the exit of the recorder programs is the same.

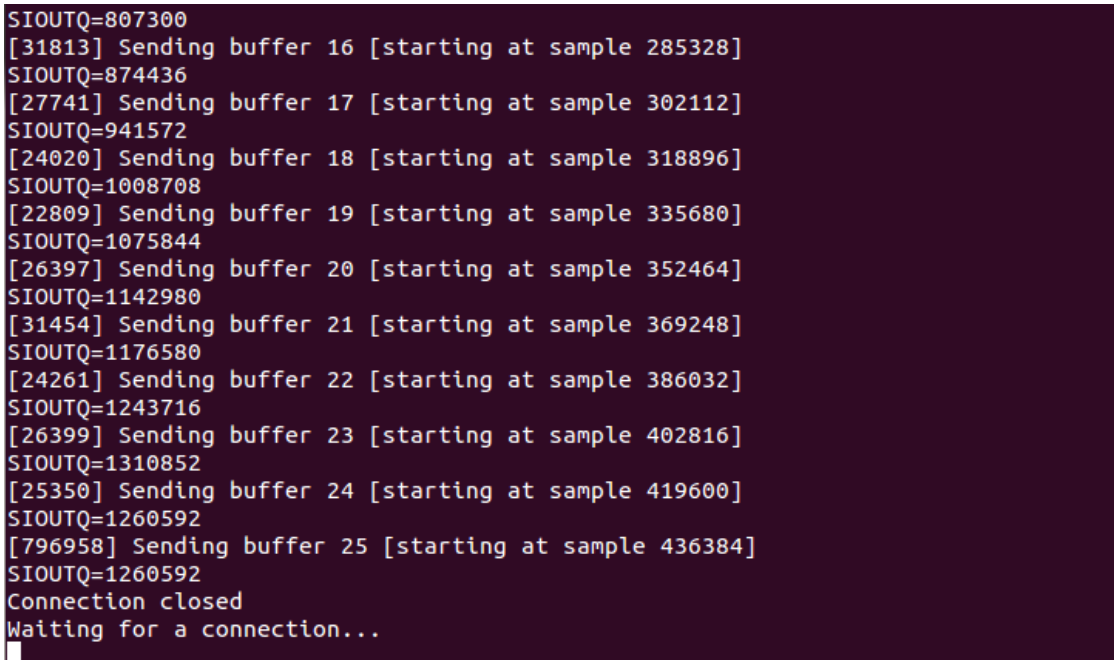

Figure 33. Recorder programs interface while sending audio over the network

Figure 34. Mixer program receiving samples for the recorder programs

8.7 PROGRAM 7 (USING A MICROPHONE AS THE INPUT)

This program is the same as program 5 but, instead of using a file to record the sound, it uses the sound recorded from a microphone to get the data. It also uses sockets to connect the recorder and the player. It is also divided in two programs/processes. The first reads data from the microphone, then prepares the sound obtained to be sent, opens a socket and waits for a connection (server). The second establishes a connection with the server (client) gets the sound from the server and reproduces it in the speakers.

- ***Experiments:*** Now, the experiments of this program are the same as the experiments of the program 5 (which uses a file) but now, the lines of code calculate the latency produced to manage both the speakers' device and the microphone device. Thus, a person is needed to speak or reproduce music in the microphone in the recorder device and another in the mixer device to play this sound and observe the results.

- ***Results:*** The result obtained was the listen of the music/voice produced by the person in the speakers with a slightly delay.

  This figures bellow shows the hardware latency between recording + playing single channel (using **pa_simple_get_latency()** function). As we can see, the recorder program (server) also works faster than the player program (client), having received only four buffers in spite of the recorder have sent six buffers.

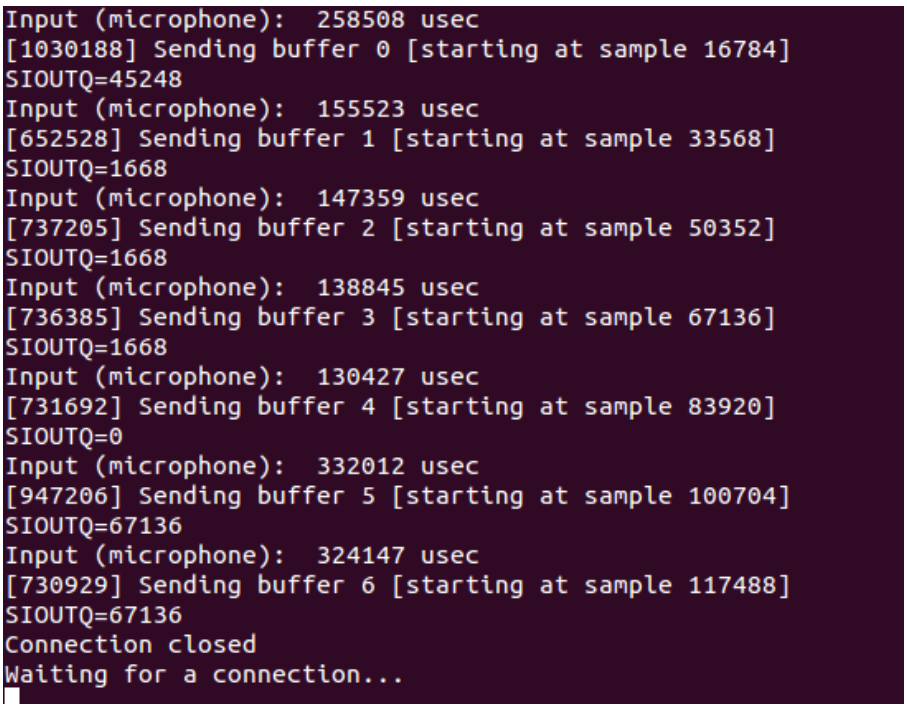

Figure 35. Recorder program sending audio over the network

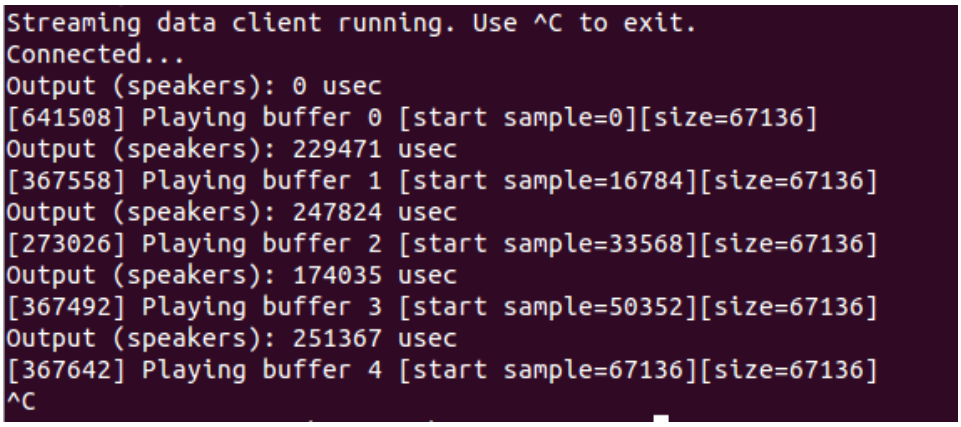

Figure 36. Player program connected with the recorder and receiving the sound buffer by the time it is reproducing it

8.8 PROGRAM 8 (USING A MICROPHONE AS THE INPUT)

This program is the same that the previous program, but it divides the record process in two processes, having two programs for the recorder part (two servers) and another one that establishes a connection with both servers (client), gets both sounds, mixes it and reproduces it in the speakers.

- ***Experiments:*** The experiments done for this program were the same as in the previous program, but here two recorder programs were needed, so a person is needed in each device to reproduce music or their voices to be sent to the third device. Thus, another person is needed in the third device to just connect it to the recorders and to start to reproduce it in the speakers and to observe the results.

- ***Results:*** The result obtained was the listen of the sound produced by the person in the speakers with a slightly delay again.

  This figures bellow shows the hardware latency between recording + playing single channel (using **pa_simple_get_latency()** function). As we can see, the recorder programs (server) works faster than the player program (client) having received only four buffers of each server in spite of the recorder have sent twenty-five buffers. Obviously the exit of the recorder programs is the same.

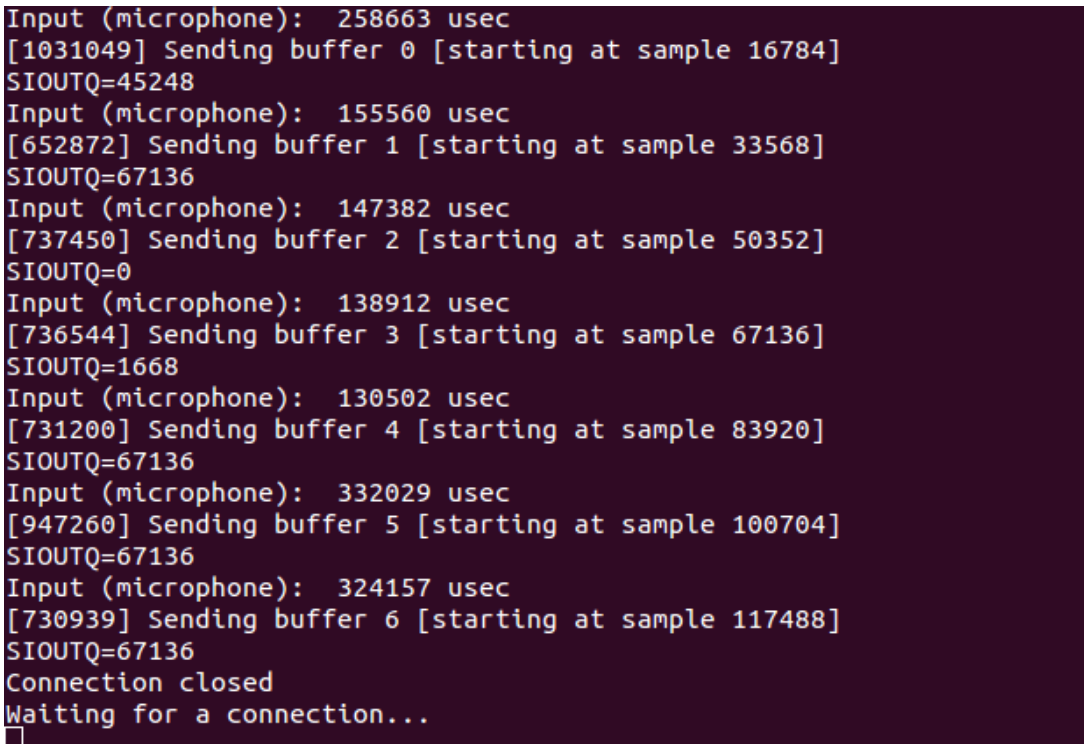

Figure 37. Recorder programs interface while sending audio over the network

```
Connected...
Output (speakers): 0 usec
[1037387] Playing buffer1 0 [start sample=0][size=67136]
[1037387] Playing buffer2 0 [start sample=0][size=67136]
Output (speakers): 129850 usec
[652102] Playing buffer1 1 [start sample=16784][size=67136]
[652102] Playing buffer2 1 [start sample=16784][size=67136]
Output (speakers): 152578 usec
[737451] Playing buffer1 2 [start sample=33568][size=67136]
[737451] Playing buffer2 2 [start sample=33568][size=67136]
Output (speakers): 94155 usec
[740806] Playing buffer1 3 [start sample=50352][size=67136]
[740806] Playing buffer2 3 [start sample=50352][size=67136]
Output (speakers): 142643 usec
[726943] Playing buffer1 4 [start sample=67136][size=67136]
[726943] Playing buffer2 4 [start sample=67136][size=67136]
^C
```

Figure 38. Mixer program receiving samples for the recorder programs

CHAPTER 9 DISCUSSION AND CONCLUSION

In this last chapter, a discuss about the project speaking about how well the solution proposed in previous chapters solves the problem, how well I addressed it, what skills I have learnt by the time I was doing it and what kind of enhancements are necessary for a future work are going to be described, as well as a little conclusion with a brief of the project and the solution.

9.1 SOLUTION REVIEW

The solution proposed that satisfies the problem is to add an offset to the mix function to combat the different latencies between the channels.

As it was shown in the chapter 8, the slightly and inevitable hardware latency was present in every program, but it was more than acceptable to hear the sound synchronized.

The latency in the TCP network was minimised as much as possible by the programming. It can be proven since measures of how much this latency is were done as you can see in the chapter 7.4 (using the ping command).

The latency produced by the operating system is shown also in the chapter 7.4 with the CPU latency program.

9.2 PROJECT REVIEW

Addressing the project step by step allowed me to develop an understanding of how to use PulseAudio and sockets as the project develops.

By the time the project was being done, a problem with the latency arose. When the experiment with sockets was started, a latency of about 10 seconds was obtained when the sound was played in the speakers. By experiments, I came to the conclusion that it was because I was initializing the playback stream (i.e., connecting with the sound server of PulseAudio) every time the loop was executed. Then, to deal with that, the code that does it

(shown in the 5.2 chapter) were put in the beginning of the program, connecting only once to the server having after barely latency in the end speakers.

After that, the project was addressed without problems and in a comfortable way thanks to the incremental method followed.

9.3 KEY SKILLS

By the time the project was being done, I have learnt about how to manage audio if it is going to be sent over a network and how to record it and edit it before being sent through this network (as shown in chapter 5).

Also, I have strengthened the knowledge about sockets, having learnt how to connect one client with two servers, thing that I have never done it before. Now I could connect any device to another via sockets without any problem, and what is more, I could send not only audio, also, any information required for any application (as shown in chapter 6).

To finish, I have strengthened also my programming skills in C language, feeling more confidence to address any project that needs to be programmed in this language.

9.4 FUTURE WORK

For a future, the Wi-Fi connection between the devices should be carry out as well as an important management of the latency since a Wi-Fi connection could introduce a lot of latency. Wi-Fi connections are more suitable for latency-sensitive applications such as voice or video since the retransmission of the packet lost is needed if the network is congested, having the signal duplicated and with a lot of delay. A better algorithm to fight again the latency and the long offset that the multi-track sound could introduce, needs to be studied deeply.

Also, different synchronization technologies can be carried out for next studies about this project. The devices can be synchronized before send any data letting the device involved know when they are going to send data by using UDP Broadcast.

Another option is to synchronize the data just before being processed by the mixer device in an intermediate synchronizer node as [1] propose.

The last option is to synchronize the data manually. To manually synchronise the two streams is needed to use an offset when mixing and update its value in real-time from the screen. There is an example program that gives you buttons to click at runtime to set the offset, and a function getValue() to call to get the latest value. The program is attached in the appendix.

9.5 CONCLUSION

A good synchronization of the sound is required in a time when several audio applications are being developed. When two devices are ready to send audio over a network, this multi-track sound will arrive at the third computer with an offset giving a negative effect to the listener.

This project has dealt with this offset achieving a good synchronization of the multi-track sound getting a good effect on the listener. This was achieved thanks to the division of the project into several steps having constantly a good vision of the problem, a good scalability and having controlled the latency at all times.

A lack of synchronization over c. 100µs is audible to the listener (as the experiments shows in the chapter 4).

REFERENCES

[1] Z. Guotao, M. Huadong, L. Hong, & S. Yan, "Adaptive Audio Synchronization Scheme based on Feedback Loop with Local Clock in Wireless Audio Sensor Networks," in 16th International Conference on Parallel and Distributed Systems, Beijing, 2010, pp. 66.

[2] S. Abderrahmane, C. Ken, & B. Azeddine, "Next-Generation Audio Networking Engineering for Professional Applications," in 20th Telecommunications forum TELFOR, France, 2012, pp. 1252-1254.

[3] M. Lester & J. Boley, "The Effects of Latency on Live Sound Monitoring," in 123rd Convention, New York, USA, 2007, pp. 1.

[4] G. G. Steven, "Sound Localization," in Multimedia Sensor Fusion for Intelligent Camera Control and Human-Computer Interaction, 1st ed. California, 1997, ch. 3, sec. 1.1, pp. 151.

APPENDICES (IN A ZIP FILE)

- Source Code
- Compiled executable with running instructions
- Interim Progress Reports
- Audio Files